RIASSUNTO COMPUTING INFRASTRUCTURES

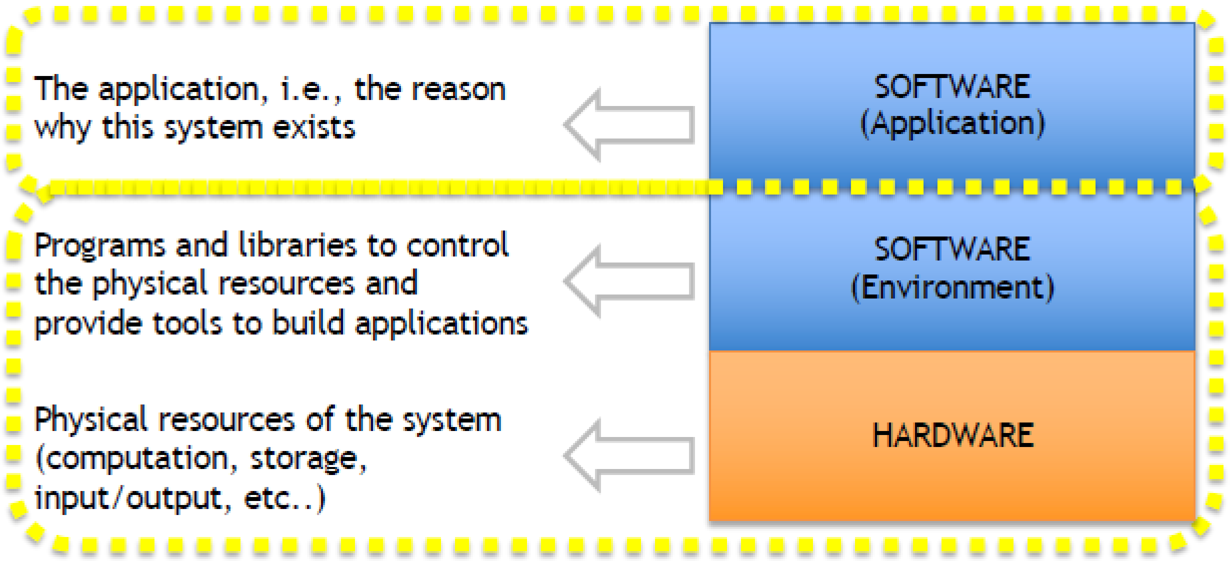
(a.a. 21/22 – written by Davide Giannubilo)

LESSON 1

What is a **computing infrastructure**?

Technological infrastructure that provides hardware and software for computation to other systems and services.

ADVANTAGES	DISADVANTAGES
Lower IT costs	Require a constant Internet connection
High performance	Do not work well with low-speed connections
Instant software updates	Hardware Features might be limited
“Unlimited” storage capacity	Privacy and security issues
Increased data reliability	High Power Consumption
Universal document access	Latency in making decision
Device independence	


The application, i.e., the reason why this system exists
SOFTWARE (Application)
Programs and libraries to control the physical resources and provide tools to build applications
SOFTWARE (Environment)
Physical resources of the system (computation, storage, input/output, etc..)
HARDWARE

LESSON_2 – DATA WAREHOUSE

Traditional enterprises are also shifting to **cloud computing** (computing and storage have moved from PC like clients to smaller, often mobile, devices, combined with large internet services).

Why this need?

- User experience improvement (ease of management, ubiquity of access).
- Advantages to vendors:
    - Software-as-a-service allows faster application development
    - Improvements and fixes in the software are easier inside their datacentres (instead of updating many millions of clients with peculiar hardware and software configurations).
    - The hardware deployment is restricted to a few well-tested configurations.
- Server-side computing allows:
    - Faster introduction of new hardware devices.
    - Many application services can run at a low cost per user.
- In addition, we have some workload that require so much computing capability that they are a more natural fit in datacentre like Search services and Machine and Deep Learning.

The trends toward server-side computing and widespread internet services created a new class of computing systems: **warehouse-scale computers (WSCs)**.

The program in warehouse-scale computing:

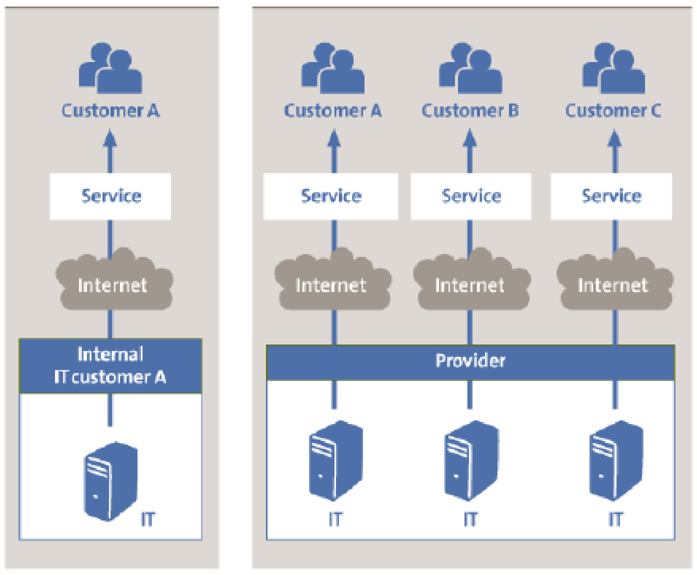
- is an internet service
- may consist of tens or more individual programs
- such programs interact to implement complex end-user services such as email, search, maps or machine learning.

Datacentre

It is a building where multiple servers and communication units are co-located because of their common environmental requirements and physical security needs, and for ease of maintenance. Traditional datacentres:

- typically host a large number of relatively small- or medium sized applications
- each application is running on a dedicated hardware infrastructure that is de-coupled and protected from other systems in the same facility
- applications tend not to communicate each other

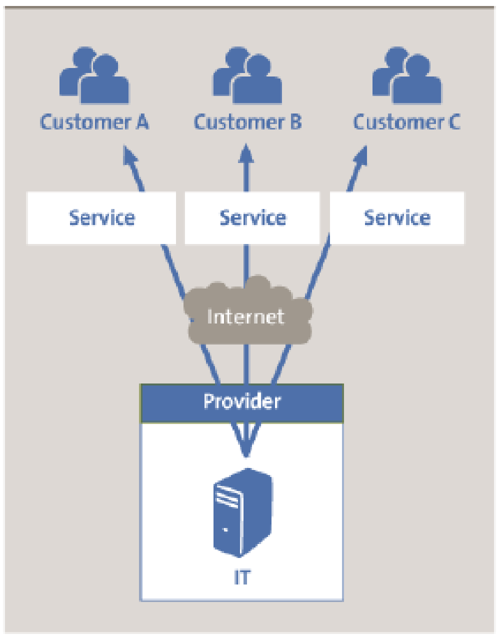
Those datacentres host hardware and software for multiple organizational units or even different companies.



WSCs

They belong to a single organization, use a relatively homogeneous hardware and system software platform, and share a common systems management layer.

- WSCs run a smaller number of very large applications (or internet services)
- The common resource management infrastructure allows significant deployment flexibility
- The requirements of homogeneity, single-organization control, cost efficiency motivate designers to take new approaches in designing WSCs.



WSCs are not just a collection of servers.
The software running on these systems executes on clusters of hundreds to thousands of individual servers (far beyond a single machine or a single rack).
The machine is itself this large cluster or aggregation of servers and needs to be considered as a single computing unit.

Multiple datacentres? Yes, they are replicas of same service:

- To improve serving throughput
- To reduce user latency

Anyway, a request is typically fully processed within one datacentre.

Services provided through WSCs must guarantee **high availability**, typically aiming for at least 99.99% uptime (i.e., one-hour downtime per year). Achieving such fault-free operation is difficult when a large collection of hardware and system software is involved.
WSC workloads must be designed to gracefully tolerate large numbers of component faults with little or no impact on service level performance and availability.
WSC has other important components related to power delivery, cooling, and building infrastructure that also need to be considered.

LESSON_3 – SERVER

Servers hosted in individual shelves are the basic building blocks of WSCs. They are interconnected by hierarchies of networks and supported by the shared power and cooling infrastructure.

Form factor that allows to fit them into the shelves:

1. Rack (1U = 44.45 mm or more)
2. Blade enclosure format
3. Tower

Servers are built in a tray or blade housing:

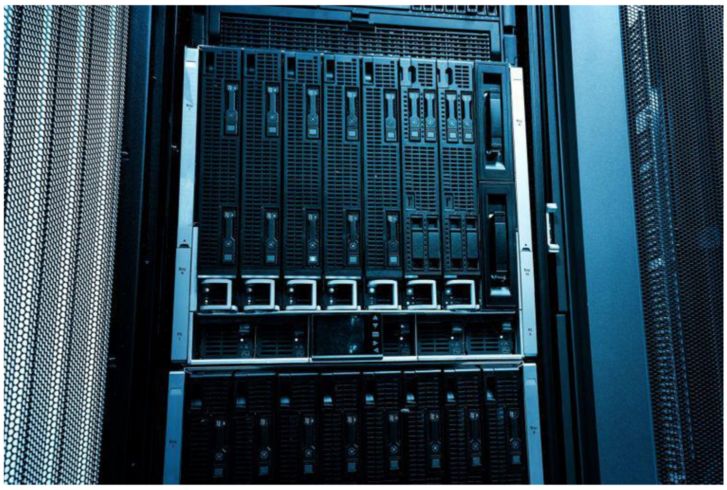
- Motherboard
- Chipset
- Additional plug-in components

Rack servers: are special shelves that accommodate all the IT equipment and allow their interconnection.
The advantage of using these racks is that it allows designers to stack up other electronic devices along with the servers. They include power delivery, battery backup, power conversion and network equipment (switches, firewall, routers).

TOWER	
PROs	**CONs**
• Scalability and ease of upgrade • Cost-effective (cheapest servers) • Cools easily (low component density)	• Consumes a lot of space • Provides a basic level of performance • Complicated cable management

RACK	
PROs	**CONs**
• Failure containment • Simplified cable management • Cost-effective (computing power and efficiency at relatively lower costs)	• Power usage (additional cooling systems due to their high component density • Maintenance

Blade servers are the latest and the most advanced type of servers in the market. They can be termed as hybrid rack servers, in which servers are placed inside blade enclosures, forming a blade system.
The biggest advantage of blade servers is that these servers are the smallest types of servers available at this time and are great for conserving space.



BLADE	
PROs	**CONs**
• Load balancing and failover • Centralized management (you can connect all the blades through a single interface, making the maintenance and monitoring easy) • Cabling • Size and form-factor	• Expensive configuration • HVAC (they are very powerful and come with a high component density so they need special accommodations in order to don't get overheated)

How are the servers located into a datacentre?

The IT equipment is stored into corridors and organized into racks. Cold air flows from the front (cool aisle), cools down the equipment, and leave the room from the back (warm aisle).

Since 2013, deep learning models began to appear and be widely adopted, enabling specialized hardware to power a broad spectrum of machine learning solutions. To satisfy the growing compute needs for deep learning, WSCs deploy specialized accelerator hardware:

- GPU (are still relatively general-purpose devices)

    - Data-parallel computations: the same program is executed on many data elements in parallel. The scientific codes are mapped onto the matrix operations. High level languages (such as CUDA and OpenCL) are required and they are up to 1000x faster than CPU.
    - GPUs are configured with a CPU host connected to a PCIe-attached accelerator tray with multiple GPUs. They within the tray are connected using high bandwidth interconnects such as NVlink.
- TPU (ML-specific hardware)
    - Custom-built integrated circuit developed specifically for machine learning and tailored for TensorFlow (powering Google datacentres since 2015)
    - A Tensor is an n-dimensional matrix. This is the basic unit of operation in TensorFlow
    - TPUs are used for training and inference
        - TPUv1 is an inference-focused accelerator connected to the host CPU through PCIe links
        - TPUv2 (4 chip, 2 cores per chip)
            - Each Tensor core has an array for matrix computations (MXU) and a connection to high bandwidth memory (HBM) to store parameters and intermediate values during computation
            - In a rack multiple TPUv2 accelerator boards are connected through a custom high-bandwidth network to provide 11.5 petaflops of ML compute. The high bandwidth network enables fast parameter reconciliation with well-controlled tail latencies. Up to 512 total TPU cores and 4 TB of total memory in a TPU Pod (64 units)
        - TPUv3 (2.5x faster than TPUv2)
            - It is the first liquid-cooled accelerator in Google’s datacentre
            - New ML capabilities and rapid neural architecture search
            - The v3 TPU Pod provides a maximum configuration of 256 devices for a total 2048 TPU v3 cores, 100 petaflops and 32 TB of TPU memory
- FPGA
    - Array of carefully designed and interconnected digital subcircuits that efficiently implement common functions offering very high levels of flexibility. The digital subcircuits are called configurable logic blocks (CLBs).
    - VHDL and Verilog are hardware description languages that allow to “describe” hardware. HDL code is more like a schematic that uses text to introduce components and create interconnections.

	Advantages	Disadvantages
CPU	• Easy to be programmed and support any programming framework. • Fast design space exploration and run your applications.	• Most suited for simple models that do not take long to train and for small models with small training set.
GPU	• Ideal for applications in which data need to be processed in parallel like the pixels of images or videos.	• Programmed in languages like CUDA and OpenCL and therefore provide limited flexibility compared to CPUs.
TPU	• Very fast at performing dense vector and matrix computations and are specialized on running very fast ML workloads	• For applications and models based on TensorFlow/PyTorch/ JAX • Lower flexibility compared to CPUs and GPUs
FPGA	• Higher performance, lower cost and lower power consumption compared to other options like CPUs and GPU	• Programmed using OpenCL and High-level Synthesis (HLS) . • Limited flexibility compared to other platforms.

LESSON_4 – STORAGE

HDD (Hard Disk Drive)

A **hard disk drive** is a data storage using rotating disks (platters) coated with magnetic material. Data is read in a random-access manner, meaning individual blocks of data can be stored or retrieved in any order rather than sequentially.
An HDD consists of one or more rigid ("hard") rotating disks (platters) with magnetic heads arranged on a moving actuator arm to read and write data to the surfaces.

Disk characteristics

- Read/write heads:
  - float on a film air (tens of nanometres) above the platters
  - one head for each magnetic platter
  - cylinder: set of tracks with the same radius
  - seek time: time required to reach the track that contains the data, 3÷14 ms.
- Diameter: about 9 cm – two surfaces
- Rotation speed: 7200 up to 15000 rpm
- Track density: 16000 TPI (Track Per Inch)
- Sectors: 512 bytes (usually) but might be different (they are numbered sequentially, have a header and an error correction)
- Heads: can be parked close to the centre or to the outer diameter (mobile drives)
- Disk buffer cache: embedded memory in a hard disk drive that has the function of a buffer between the disk and the computer (several MB)

Disk service & response time

- service time s_{disk}: seek time + rotational latency + data transfer time + controller overhead
  - seek time = head movement time (ms), is a function of the number of cylinders traversed
  - latency time = time to wait for the sector (ms, $1/2$ round)
  - transfer time = is a function of rotation speed, storing density, cylinder position (MB/sec)
  - controller overhead = buffer management (data transfer) and interrupt sending time (no queue, average time to serve a single I/O request)
- response time: service time + queue time (average time to serve an I/O request in working conditions)

SSD (Solid State Drive)

- No mechanical or moving parts like HDD
- Built out of transistors (like memory and processors)
- Retain information despite power loss unlike typical RAM
- A controller is included in the device with one or more solid state memory components
- It uses traditional hard disk drive (HDD) interfaces (protocol and physical connectors) and form factors
- Higher performance than HDD

How do they store bits?

1. Single-level cell (SLC) – single bit per cell
2. Multi-level cell (MLC) – two bits per cell
3. Triple-level cell (TLC) – three bits per cell
4. QLC, PLC, …

NAND flash is organized into Pages and Blocks

- A **page** contains multiple logical block (e.g. 512B-4KB) addresses (LBAs).
  It is the smallest unit that can be read/written. It is a sub-unit of an erase block and consists of the number of bytes which can be read/written in a single operation through the READ or PROGRAM commands.
- A **block** typically consists of multiple pages (e.g., 64) with a total capacity of around 128-256KB.
  It is the smallest unit that can be erased and consists of multiple pages and can be cleaned using the ERASE command.
- **Block/Page** terminology in the SSD context can clash with previous use

(I can write a page only when it has been erased; to change a content of a page, we have to erase the full block and after we can change the content of the page)

Pages can be in 3 states:

- Dirty or INVALID (they contain data, but this data is no longer in use (or never used))
- Empty or ERASED
- In use or VALID

Only empty pages can be written.
Only dirty pages can be erased, but this must be done at the block level (all the pages in the block must be dirty or empty).
If no empty page exists, some dirty page must be erased

- If no block containing just dirty or empty pages exists, then special procedures should be followed to gather empty pages over the disk
- To erase the value in flash memory the original voltage must be reset to neutral before a new voltage can be applied, known as **write amplification**. (Write amplification: the actual amount of information physically written to the storage media is a multiple of the logical amount intended to be written)

Access speed → 15-20x faster than HDD
Read speed → more or less are equal – Why?

Wear out: breakdown of the oxide layer within the floating-gate transistors of NAND flash memory. The erasing process hits the flash cell with a relatively large charge of electrical energy. Each time a block is erased:

- the large electrical charge actually degrades the silicon material

- after enough write-erase cycles, the electrical properties of the flash cell begin to break down and the cell becomes unreliable.

Flash Translation Layer (key component): is an SSD component that make the SSD "looks like HDD" because a direct mapping between Logical to Physical pages is not feasible.

- Data Allocation and Address translation
    - Efficient to reduce Write Amplification effects
    - Program pages within an erased block in order (from low to high pages)
        - Log-Structured FTL
- Garbage collection (GC)
    - Reuse of pages with old data (Dirty/Invalid)
- Wear levelling
    - FTL should try to spread writes across the blocks of the flash ensuring that all of the blocks of the device wear out at roughly the same time

Garbage collection: is the process of finding garbage blocks and reclaiming them.
When an existing page is updated → old data becomes obsolete!
Old version of data are called garbage and (sooner or later) garbage pages must be reclaimed for new writes to take place.

This procedure is expensive because it requires reading and rewriting of live data and it depends on the amount of data block that have to be migrated.
(Ideal garbage collection is reclamation of a block that consists of only dead pages)

Solution to alleviate the problem:

- Overprovision the device by adding extra flash capacity
- Run the Garbage Collection in the background using less busy periods for the disk

When performing background GC the SSD assumes to know which pages are invalid.

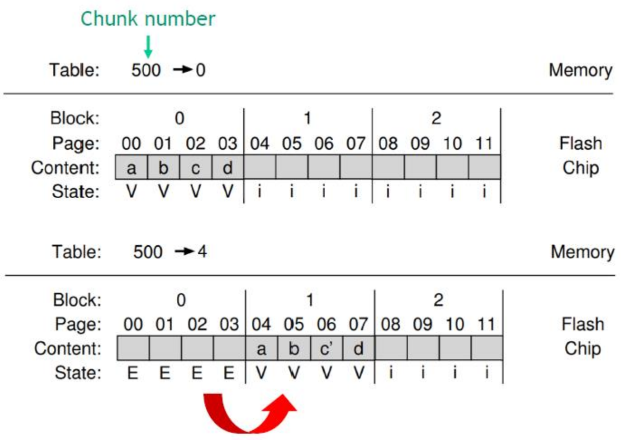
Problem: most file systems don't delete data (ex. Linux). Lack of explicit delete means the GC wastes effort copying useless pages.

To solve this → new SATA command TRIM (SCSI – UNMAP)
The OS tells the SSD that specific LBAs (Logical Block Addressing) are invalid and may be GCed (OSs like Linux, Win7)

Mapping Table Size: is too large, e.g. with a 1TB SSD with a 4byte entry per 4KB page, 1GB of is needed for mapping.
Approaches to reduce the size:

- Block-based mapping
    - Coarser grain approach
        - Mapping at block granularity but there is a small problem: the FTL must read a large amount of live data from the old block and copy them into a new one.

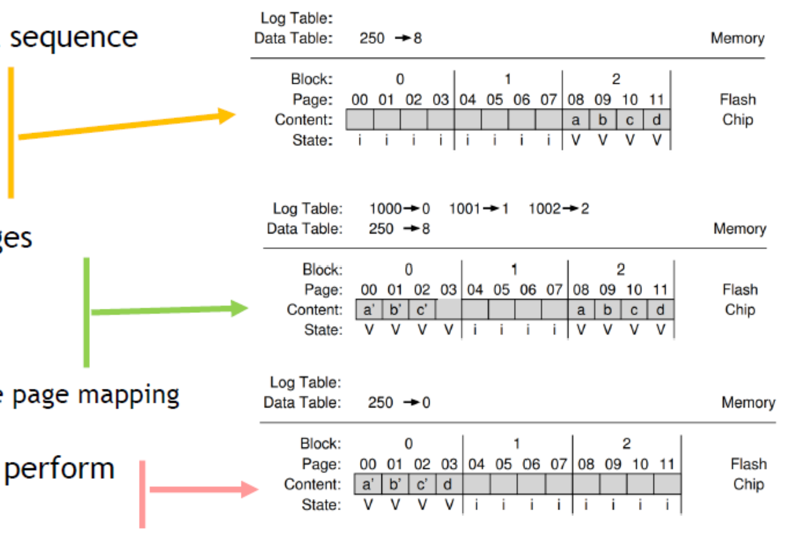
- **Example:**
  - Write(2000, a)
  - Write(2001, b)
  - Write(2002, c)
  - Write(2003, d)
  - Write(2002, c')


Chunk number
Table: 500 → 0
Memory
Block: 0 1 2
Page: 00 01 02 03 04 05 06 07 08 09 10 11
Content: a b c d
State: V V V V i i i i i i i i
Flash Chip
Table: 500 → 4
Memory
Block: 0 1 2
Page: 00 01 02 03 04 05 06 07 08 09 10 11
Content: a b c' d
State: E E E E V V V V i i i i
Flash Chip

- Hybrid mapping
  - Multiple tables
    - 2 tables: Log blocks & Data blocks

When looking for a particular logical block, the FTL will consult the page mapping table and block mapping table in order.

- Example:
  - Let's suppose the past sequence
    - Write(1000, a)
    - Write(1001, b)
    - Write(1002, c)
    - Write(1003, d)
  - Let's update some pages
    - Write(1000, a')
    - Write(1001, b')
    - Write(1002, c')
    - FTL updates only the page mapping information
  - When needed FTL can perform MERGE operations


Log Table:
Data Table: 250 → 8
Memory
Block: 0 1 2
Page: 00 01 02 03 04 05 06 07 08 09 10 11
Content: a b c d
State: i i i i i i i i V V V V
Flash Chip
Log Table: 1000 → 0 1001 → 1 1002 → 2
Data Table: 250 → 8
Memory
Block: 0 1 2
Page: 00 01 02 03 04 05 06 07 08 09 10 11
Content: a' b' c' a b c d
State: V V V V i i i i V V V V
Flash Chip
Log Table:
Data Table: 250 → 0
Memory
Block: 0 1 2
Page: 00 01 02 03 04 05 06 07 08 09 10 11
Content: a' b' c' d
State: V V V V i i i i i i i i
Flash Chip

- Page mapping plus caching
  - Exploiting Data Locality
    - Basic idea is to cache the active part of the page-mapped FTL
      If a given workload only accesses a small set of pages, the translations of those pages will be stored in the FTL memory.
      High performance without high memory cost if the cache can contain the necessary working set. Cache miss overhead exists.

Wear levelling: Erase/Write cycle is limited in Flash memory. Skewness in the EW cycles shorten the life of the SSD.
Log-Structured approach and garbage collection helps in spreading writes. However, a block may consist of cold data.

- The FTL must periodically read all the live data out of such blocks and re-write it elsewhere.
- Wear levelling increases the write amplification of the SSD and decreases performance

    - Simple Policy: each Flash Block has EW cycle counter. Maintain
      $$|\max(EW\ cycle) - \min(EW\ cycle)| < e$$

SSD are not affected by data-locality and **must not be defragmented**.

HDD vs SSD

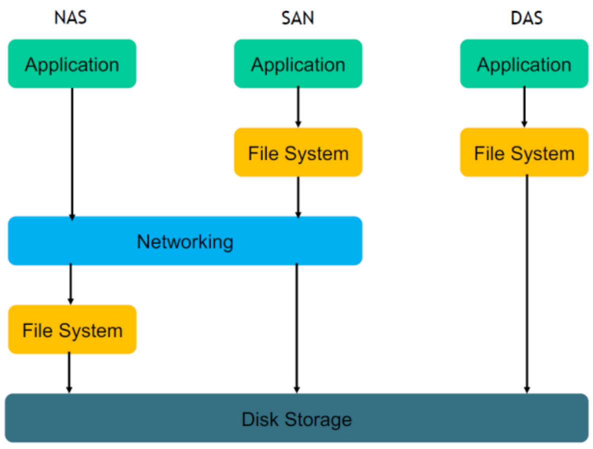
- Unrecoverable Bit Error Ratio (UBER): A metric for the rate of occurrence of data errors, equal to the number of data errors per bits read
- Endurance rating: Terabytes Written (TBW is the total amount of data that can be written into an SSD before it is likely to fail). The number of terabytes that may be written to the SSD while still meeting the requirements

A typical TBW for a 250 GB SSD is between 60 and 150 Terabytes of data written to the drive. (This means that in order to overcome, for example, a TBW of 70 Terabytes, a user should write 190 GB every day for a year or fill his SSD on a daily basis for two thirds with new files for a whole year)
It is difficult to comment on the duration of SSDs. (Dell says between 3 months and 10 years, however, depends on a lot of factors like temperature and workload)

Hybrid solutions: HDD + SDD
Some large storage servers use SSD as a cache for several HDD. Some mainboards of the latest generation have the same feature: they combine a small SSD with a large HDD to have a faster disk.
Some HDD manufacturers produce Solid State Hybrid Disks (SSHD) that combine a small SDD with a large HDD in a single unit.

Storage Systems

1. **Direct Attached Storage (DAS)** is a storage system directly attached to a server or workstation. They are visible as disks/volumes by the client OS
2. **Network Attached Storage (NAS)** is a computer connected to a network that provides only file-based data storage services (e.g., FTP, Network File System and SAMBA) to other devices on the network and is visible as File Server to the client OS
3. **Storage Area Networks (SAN)** are remote storage units that are connected to a server using a specific networking technology (e.g., Fiber Channel) and are visible as disks/volumes by the client OS



DAS

It is a storage system directly attached to a server or workstation.

Features:

- Limited scalability
- Complex management
- To read files in other machines, the "file sharing" protocol of the OS must be used

DAS does not mean "internal drives". All the external drives connected with a point-to-point protocol to a PC can be considered as DAS.

NAS

It is a computer connected to a network that provides only file-based data storage services to other devices on the network. It contains one or more hard disks, often organized into logical redundant storage containers or RAID.

Provide file-access services to the hosts connected to a TCP/IP network though Networked File Systems/SAMBA.

- Each NAS element has its own IP address.
- Good scalability (incrementing the devices in each NAS element or incrementing the number of NAS elements)

NAS is used for low-volume access to a large amount of storage by many users.

SAN

They are remote storage units that are connected to a PC/server using a specific networking technology.

- Two distinct networks (one TCP/IP and one dedicated network, e.g., Fiber Channel, iSCSI)
- High scalability (simply increasing the storage devices connected to the SAN network)

SAN is the solution for petabytes (10^{12}) of storage and multiple, simultaneous access to files, such as streaming audio/video.

	Application Domain	Advantages	Disadvantages
DAS	• Budget constraints • Simple storage solutions	• Easy setup • Low cost • High performance	• Limited accessibility • Limited scalability • No central management and backup
NAS	• File storage and sharing • Big Data	• Scalability • Greater accessibility • Performance	• Increased LAN traffic • Performance limitations • Security and reliability
SAN	• DBMS • Virtualized environments	• Improved performance • Greater scalability • Improved availability	• Costs • Complex setup and maintenance

LESSON_5 – NETWORKING

The performance of servers increases over time and the demand for inter-server bandwidth naturally increases, so we can double the aggregate compute capacity or the aggregate storage simply by doubling the number of compute or storage elements

Networking has no straightforward horizontal scaling solution.

Doubling leaf bandwidth is easy, but double the network ports and thus we have doubled the bandwidth

But if we assume that every server needs to talk to every other server, we need to deal with bisection bandwidth.
In fact, we have to double also that one because it gives the true bandwidth available in the entire system and represents bandwidth characteristics of the network better than any other metric.

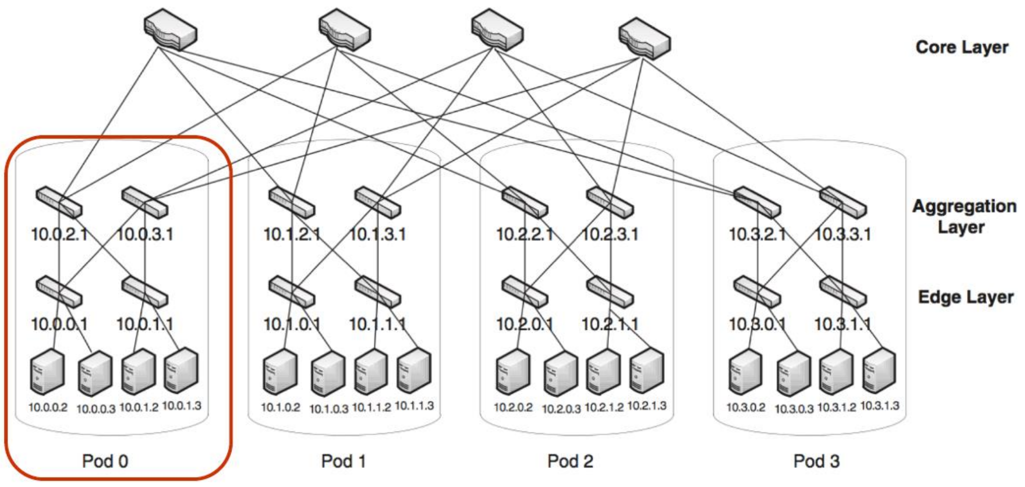
Three layers architecture reflects the topology of the datacentre:

1. Core (ex. 25 Gbit Ethernet)
2. Aggregation (Switches are positioned one per corridor, at the end of a line of rack (**EOL**). Aggregation switches must have a larger number of ports, and longer cables are required (higher costs). However, the system can scale to have a larger number of machines.
   (ex. 10 Gbit Ethernet)
3. Access (Access switches are put at the top of each rack (**TOR**). The number of cables is limited. The number of ports per switch is also limited (lower costs). However, the scalability is also limited. (ex. 1 Gbit Ethernet)

Bandwidth can be increased by increasing the switches at the core and aggregation layers, and by using routing protocols such as Equal Cost Multiple Path (ECMP) that equally shares the traffic among different routes.

The cost in term of acquisition and energy consumption can be very high. Another benefit of SANs: a way to tackle network scalability, offload some traffic to a special-purpose network connecting servers to storage units.

Fat-tree topology: it uses a larger number of slower speed switches and connections. In particular, nodes are divided into pods characterized by the same number of nodes and switches.



D-Cell topology: it defines the network in recursive way. Cells are organized in levels. Switches connects nodes at the lower level.
Some nodes belong to different cells: they perform routing among them to create a higher level cell.

Networking in High Performance Clusters

- They often have a much lower ratio of computation to network bandwidth
- Applications (e.g., weather simulations) distribute their data across RAM in all nodes, and nodes need to update neighbouring nodes after performing relatively few floating-point computations
- Traditional HPC systems have used proprietary interconnects with leading-edge link bandwidths, much lower latencies, and some form of a global address space (where the network is integrated with CPU caches and virtual addresses)
- High throughputs but much more expensive solutions

Network challenges to support Virtualization

- Connection endpoints (i.e., IP address/port combinations) can move from one physical machine to another
- Typical networking hardware as well as network management software doesn't anticipate such moves and in fact often explicitly assume that they're not possible
- Solution: the cluster manager that decides the placement of computations also updates the network state through programmable Network control planes (Software Defined Networks)

The interplay of storage and networking technology

The success of WSC distributed storage systems can be partially attributed to the evolution of datacentre networking fabrics:

- disk locality is no longer relevant in intra-datacentre computations.

This enables:

1. simplifications in the design of distributed disk-based storage systems

2. utilization improvements

Computer architects are trained to solve the problem of finding the right combination of performance and capacity from the various building blocks that make up a WSC.

Three important considerations:

1. Smart programmers may be able to restructure their algorithms to better match a more inexpensive design alternative
2. The most cost-efficient and balanced configuration for the hardware may be a match with the combined resource requirements of multiple workloads
3. Fungible resources tend to be more efficiently used

A large application that requires servers in many racks to operate must deal effectively with large discrepancies in latency, bandwidth, and capacity.

- A key challenge for architects of WSCs is to smooth out these discrepancies in a cost-efficient manner
- A key challenge for software architects is to build SW infrastructure and services that hide this complexity

LESSON_6 – BUILDING and INFRASTRUCTURES

WSC has other important components related to power delivery, cooling, and building infrastructure that also need to be considered.

In order to protect against power failure, battery and diesel generators are used to backup the external supply.

The UPS typically combines three functions in one system:

1. contains a transfer switch that chooses the active power input (either utility power or generator power);
2. contains some form of energy storage (electrical, chemical, or mechanical) to bridge the time between the utility failure and the availability of generator power
3. conditions the incoming power feed, removing voltage spikes or sags, or harmonic distortions in the AC feed.

The cooling system is usually a very expensive component of the datacentre, and it is composed by coolers, heat-exchangers, and cold-water tanks.

Various topology:

- **Open-Loop**, the simplest topology is fresh air cooling (or air economization) – essentially, opening the windows.
  It refers to the use of cold outside air to either help the production of chilled water or directly cool servers. It is not completely free in the sense of zero cost, but it involves very low-energy costs compared to chillers.
- **Closed-Loop**, there are many forms but the most common is the one where the goal is to isolate and remove heat from the servers and transport it to a heat exchanger.
  Cold air flows to the servers, heats up, reaches the heat exchanger and so on in this way.
  - **Closed-Loop with 2 loops**, cold air from the underfloor plenum to racks and the warm air goes into the "CRAC units" that discharge the heat and new cold air goes underfloor and so on.
  - **Closed-Loop with 3 loops**, it uses heat exchangers like evaporators and condensers in the second loop and evaporating cooling towers into the third loop.

Trade-offs:

- Fresh air cooling can be very efficient but does not work in all climates, requires filtering of airborne particulates, and can introduce complex control problems
- Two-loop systems are easy to implement and offer isolation from external contamination, but typically have lower operational efficiency
- A three-loop system is the most expensive to construct and has moderately complex controls but offers contaminant protection and good efficiency.

New cooling systems:

- **In-rack cooler**, it adds an air-to-water heat exchanger at the back of a rack so the hot air exiting the servers immediately flows over coils cooled by water, essentially reducing the path between server exhaust and CRAC input.

- **In-row cooling**, it works like in-rack cooling except the cooling coils are not in the rack, but adjacent to the rack.
- **Liquid cooling**, using cold plates. The liquid circulating through the heat sinks transports the heat to a liquid-to-air or liquid-to-liquid heat exchanger that can be placed close to the tray or rack or be part of the datacentre building (such as a cooling tower).

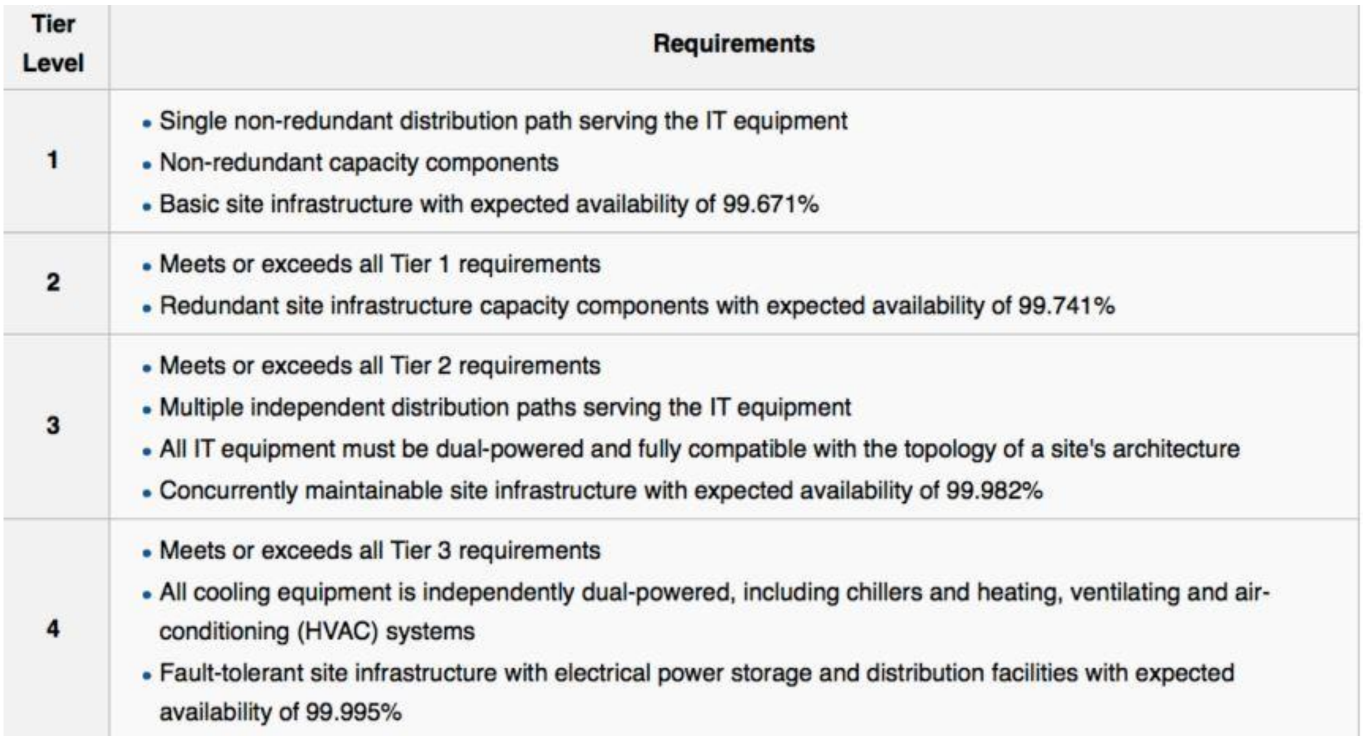
Power consumption:

- Cooling usually requires about half the energy required by the IT equipment (servers + network + disks).
- Energy transformation creates also a large amount of energy wasted for running a datacentre.
- Power usage effectiveness (PUE) is the ratio of the total amount of energy used by a DC facility to the energy delivered to the computing equipment

$$PUE = \frac{\text{Total facility Power}}{IT\ Equipment\ Power}$$

(1 è il valore ideale)

Data-centre availability is defined by in four different tier level.



Tier Level	Requirements
1	• Single non-redundant distribution path serving the IT equipment • Non-redundant capacity components • Basic site infrastructure with expected availability of 99.671%
2	• Meets or exceeds all Tier 1 requirements • Redundant site infrastructure capacity components with expected availability of 99.741%
3	• Meets or exceeds all Tier 2 requirements • Multiple independent distribution paths serving the IT equipment • All IT equipment must be dual-powered and fully compatible with the topology of a site's architecture • Concurrently maintainable site infrastructure with expected availability of 99.982%
4	• Meets or exceeds all Tier 3 requirements • All cooling equipment is independently dual-powered, including chillers and heating, ventilating and air-conditioning (HVAC) systems • Fault-tolerant site infrastructure with electrical power storage and distribution facilities with expected availability of 99.995%

LESSON_7 – VIRTUALIZATION

Hardware resources (CPU, RAM, etc.) are partitioned and shared among multiple virtual machines (VMs).
The virtual machine monitor (VMM) governs the access to the physical resources among running VMs. (Performance, isolation, and security)

In computer architecture, the set of instructions that a program can use might be structured at different levels.
The ISA corresponds to Level 2 in the layered execution model. ISA marks the division between hardware and software.

User ISA	System ISA
Aspects of the ISA that are visible to an application program (sum, multiplication, logical operations, branches, etc...). When application interacts with the HW, User ISA is used.	Aspects visible to supervisor software (i.e., the OS) which is responsible for managing hardware resources (e.g., hide the complexity of CPUs, define how app access memory, communicate with the HW). When the OS interacts with the HW (Drivers, MM, Sched.), System ISA is used.

The ABI corresponds to Level 3 in the layered execution model. (Abstraction Binary Interface)
One machine level can only run instructions that were meant for it.

Virtual Machine (VM): is a logical abstraction able to provide a virtualized execution environment.

- (provides) Identical Software behaviour
- (consists in a) Combination of physical machine and virtualizing software
- (may appear as) Different resources than physical machine
- (may result in) Different level of performance

Its tasks are:

- To map virtual resources or states to corresponding physical ones
- To use physical machine instructions/calls to execute the virtual ones

There are two types of Virtual Machine, and we need to introduce two concepts:

Host → the underlying platform supporting the environment/system.
Guest → the software that runs in the VM environment as the guest.

System VMs

- Provide a complete system environment that can support an operating system (potentially with many user processes)
- It provides operating system running in it access to underlying hardware resources (networking, I/O, a GUI).
- The VM supports the operating system as long as the system environment is alive.

- Virtualizing software placed between hardware and software (emulates the ISA interface seen by software)
- The virtualization software is called VMM (Virtual Machine Monitor). The VMM can provide its functionality either working directly on the hardware or running on another OS.

Classic-System VM

- The VMM is on bare hardware, and virtual machines fit on top
- The VMM can intercept guest OS's interaction with hardware resources
- The most efficient VM architecture (HW executes the same instructions of VMs)
- Two different OSs on the same HW

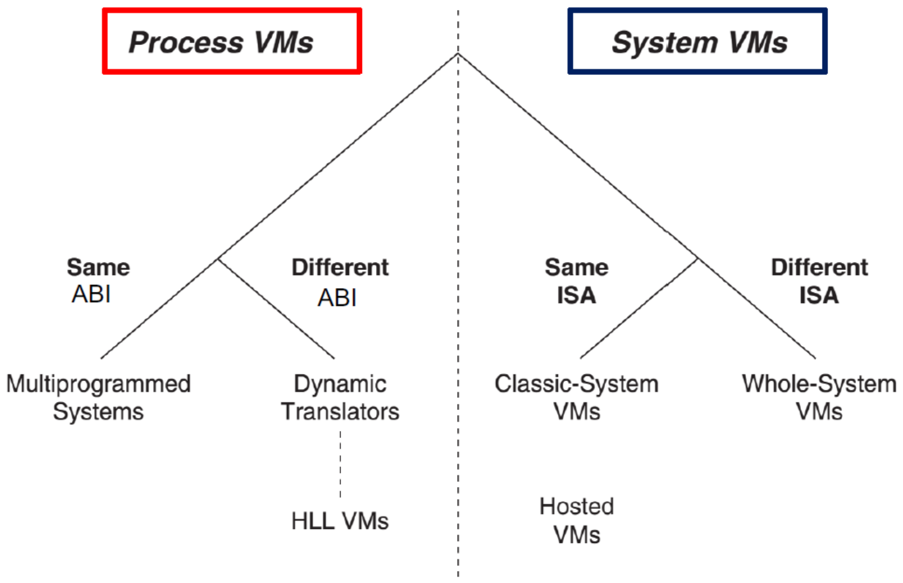
Hosted VM: virtualizing software is on top of an existing host operating system.

Whole-system VMs

- Virtualize all software: ISAs are different so both application and OS code require emulation, e.g., via binary translation. (no native execution possible)
- Usually implement the VMM and guest software on top of a conventional host OS running on the hardware.
- The VM software must emulate the entire hardware environment and all the guest ISA operations to equivalent OS call to the Host. (ex. MS Word and Windows running on a Power PC (not x86 family))

Process VMs

- The runtime software supports the levels 0-3 of the architecture
- Able to support an individual process
- The virtualizing software is placed at the ABI interface, on top of the OS/hardware combination
- The virtualizing software emulates both user-level instructions and operating system calls (this software is usually called Runtime Software)



Multiprogrammed systems

- VM formed by OS call interface + user ISA
- Common approach of all modern OS for multiuser support
    - Task/Process Manager
- Each user process is given:
    - the illusion of having a complete machine to itself
    - its own address space and is given access to a file structure
- OS timeshares and manages HW resources to permit this

Emulation (Different ABI / Different ISA)

- It refers to those software technologies developed to allow an application (or OS) to run in an environment different from that originally intended.
- It reads all the bytes included in the memory of system it is going to reproduce with an interpreter. (Can be a slow process)

High Level Language VM (ex. Java Virtual Machine that also need an interpreter called JRE)

- The goal is to have an isolated execution environment for each application.
- VM Task:
    - Translates application byte code to OS-specific executable.
    - Minimize HW/OS-specific feature for platform independence
- Applications run normally but
    - Sandboxed
    - Can “migrate”
    - Are not conflicting one another
    - Are not “installed” in a strict sense

Types of virtualization

Depending on where the new layer is placed, we obtain different types of virtualizations.

1. **Hardware level virtualization**, it is placed between hardware and OS. The interface seen by OS and application might be different from the physical one.
2. **Application-level virtualization**, a virtualization layer is placed between the OS and some applications (Java Virtual Machine). Applications run in their environment, independently from OS.
3. **System-level virtualization**, the virtualization layer provides the interface of a physical machine to a secondary OS and a set of application running in it, allowing them to run on top of an existing OS (VMware or VirtualBox)

Virtualization properties:

- Partitioning (multiple OSs on a single physical machine and partitioning of resources between different VMs)
- Isolation (Fault tolerance, security and advanced resource control to guarantee performance)
- Encapsulation (VM can be saved in a file and we can copy or move it)

- HW-independence (Provisioning/migration of a given VM on a given physical server)

Virtual Machine Managers (System VMs – Same ISA)

It is an application that:

- manages the VMs
- mediates access to the hardware resources on the physical host
- intercepts and handles any privileged or protected instructions issued by the virtual machines.

This type of virtualization typically runs virtual machines whose operating system, libraries, and utilities have been compiled for the same type of processor and instruction set as the physical machine on which the virtual systems are running.

Three terms for the same thing:

- Virtual Machine Manager – It provides a user-friendly interface to create, configure and maintain virtualized resources.
- Virtual Machine Monitor – The virtualization software.
- Hypervisor – Virtualization software that runs directly on the hardware
    - **Type 1**, takes direct control of the hardware
        i. **Monolithic architecture**, device drivers run within the hypervisor, so better performance and isolation but it can run only on hardware for which the hypervisor has drivers
        ii. **Microkernel architecture**, device drivers run within a service VM, so smaller hypervisor and we can use 3rd party drivers
    - **Type 2**, reside within a host operating system
        i. At least 2 OSs running on the same hardware (Host OS & Guest OS). The VMM runs in the Host OS.
           It is more flexible in terms of underlying hardware, and it is simple to manage and configure but we need to avoid conflict between Host OS and Guest OS and the Host OS consumes a set of physical resources.

Virtualization techniques (System VMs – Same ISA)

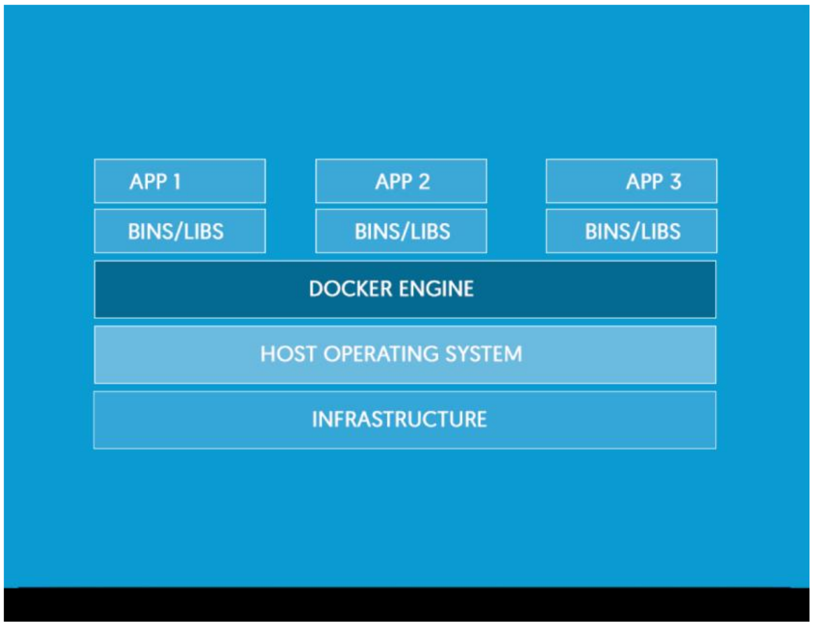
1. Paravirtualization
    a. VMM present to VMs an interface similar but not identical to that of the underlying hardware
    b. The goal is to reduce guest's executions of tasks too expensive for the virtualized environment
    c. Simpler VMM and high performance but the Guest OS is modified
2. Full virtualization
    a. It provides a complete simulation of the underlying hardware
    b. The pro is that the Guest OS is unmodified but to allow guest's task and request is requested the mediation of the Hypervisor and this virtualization technique does not work on every architecture.

Container

Containers are pre-configured packages with everything you need to execute the code in the target machine.

The main advantage of containers is that their behaviour is predictable, repeatable, and immutable: when I create a "*master*" container and duplicate it on another server, I know exactly how it will be executed. There are no unexpected errors when moving it to a new machine or between environments.

The main difference is that the containers share the host system kernel with other containers.



Containers ease the deployment of applications and increase the scalability, but they also impose a modular application development where the modules are independent and uncoupled.
Also useful to create multi-user *Platform-as-a-Service* (PaaS) or *Software-as-a-Service* (SaaS).

Docker

It aims to create and distribute software inside containers. Open-source platform of tools that helps to "build, ship and run any app, anywhere. It is based on a Docker File (text file) and it is possible to run the docker where the docker daemon is running.

Kubernetes

It is for the management of medium and large clusters and complex applications.
Extensive and customizable solution that allows you to coordinate large-scale node clusters in production more efficiently:

- Run containers on many heterogeneous machines
- Increase or decrease the performance by proportionally modifying (adding or even removing) the containers
- Share the load between containers
- Start new containers on different machines if a machine fails

LESSON_8 – Cloud Computing

It is a model for enabling convenient and on-demand network access to a shared pool of configurable computing resources, like networks, server, storage, applications, etc. that can be rapidly provisioned and released with minimal management effort or service provider interaction.

Cloud application (SaaS): users access the services provided by this layer through web-portals and are sometimes required to pay fees to use them. Cloud applications can be developed on the cloud software environments or infrastructure components. (ex. Gmail)

Cloud Software Environment (PaaS): users are application developers (ex. Amazon SageMaker or Azure Machine Learning). Providers supply developers with a programming-language-level environment with well-defined an API:

- Facilitate interaction between environment and apps
- Accelerate the deployment
- Support scalability

Cloud Software Infrastructure:

- IaaS: computational (VMs)
- DaaS: storage (store and access data anytime from any place)
- CaaS: communications (VoIP or video conferencing services)

Provides resources to the higher-level layers (i.e., Software and Software Environment)

Types of cloud

Public

- available on a rental basis
- fully customer self-service
- accountability is e-commerce based.

Private (internally managed datacentres)

- the organization sets up a virtualization environment on its own servers
- you have total control over every aspect of the infrastructure
- advantages of virtualization
- some issues with the capital investment and the flexibility

Community clouds

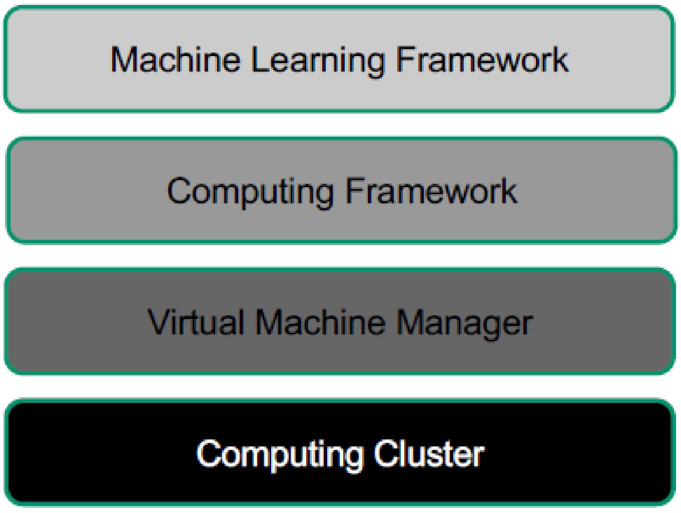
- a single cloud managed by several federated organizations
- combining several organizations allows economy of scale
- Resources can be shared and used by one organization, while the others are not using them
- like the private ones
- hosted locally or externally (typically community clouds share infrastructures of the participants)

Hybrid
It is a combination of any of the previous ones.

Cloud PROs and CONs

PROs	CONs
• Lower It costs • Improved performance • Instant software updates • "Unlimited" storage capacity • Increased data reliability • Universal document access • Device independence	• Constant Internet connection • Does not work well with low-speed connections • Feature might be limited • Stored data can be lost or might not be secure

LESSON_9 – MLaaS

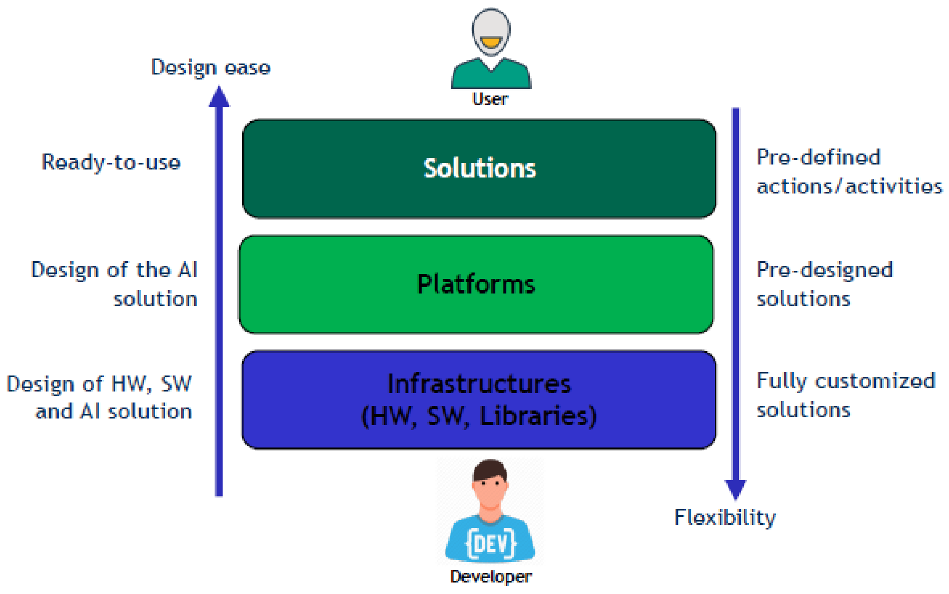


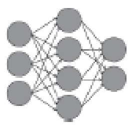
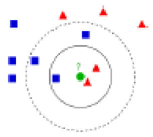
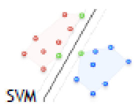
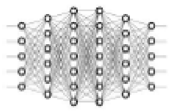
Machine learning frameworks cover a variety of learning methods for classification, regression, clustering, anomaly detection, and data preparation, and it may or may not include neural network methods.

Deep learning frameworks cover a variety of neural network topologies with many hidden layers.

Cloud computing simplifies the access to ML capabilities for

- designing a solution (without requiring a deep knowledge of ML)
- setting up a project (managing demand increases and IT solution)


Design ease
User
Ready-to-use
Solutions
Pre-defined actions/activities
Design of the AI solution
Platforms
Pre-designed solutions
Design of HW, SW and AI solution
Infrastructures (HW, SW, Libraries)
Fully customized solutions
Flexibility
{DEV}
Developer

	 Neural Networks	 K Nearest Neighbour	 SVM	 Deep Learning
Data Center	🟩	🟩	🟩	🟩
Edge Computing	🟩	🟩	🟩	🟩
Embedded PCs	🟩	🟩	🟩	🟨
IoT	🟩	🟨	🟩	

(green for training, yellow for recall)

LESSON_10 – RAID

Use multiple disks to create the illusion of a large, faster, more reliable disk.

Externally, RAID looks like a single disk but, internally, RAID is a complex computer system and disks managed by a dedicated CPU + Software.

RAID 0

Data are written on a single logical disk and split in several blocks distributed across the disks according to a striping algorithm.

- Lowest cost
- Best write performance
- Single disk failure will result in data loss

Chunk size impacts array performance. (Typically array use 64kB)

- Smaller chunks → greater parallelism
- Big chunks → reduced seek times

Capacity: N (n disks of x GB)
Reliability: 0 (any drive fails, data loss)

RAID 1

Whenever data is written to a disk it is also duplicated (mirrored) to a second disk (there are always two copies of the data), minimum 2 disk drives.

This kind of RAID is never used because of the cost.

- High reliability (if a disk fails, there is the other one)
- Read of data (shorter queue)
- Fast writes (no error) but still slower than standard disks (due to duplication)
- High costs (since 50% of capacity is used for mirroring)

Capacity: $N/2$
Reliability: 1 drive can fail. If we are lucky, $N/2$ drives can fail without data loss.

Combined RAID levels

RAID 0+1

- striping first
- then mirroring

Minimum 4 drives and after the first failure the model becomes as a RAID 0.

RAID 1+0

- mirroring first
- then striping

Minimum 4 drivers and this technique is used in databases with very high workloads.

Performance and storage capacity is the same for both. The main difference is the fault tolerance level:

- RAID 0 + 1 is less fault tolerant
- RAID 1 + 0 is larger fault tolerant

The consistency problem: mirrored writes should be atomic but what happens if there is a power failure?
Many RAID controllers include a write-ahead log

- Battery backed, non-volatile storage of pending writes
- A recovery procedure ensures to recover the out-of-sync mirrored copies

RAID 4

Disk N only stores parity information for the other N-1 disks. Parity is calculated using XOR.

Additive parity

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	0	0	1	0 ^ 0 ^ 0 ^ 1 = 1

Terrible writes performance. The bottleneck is the parity drive.

RAID 5

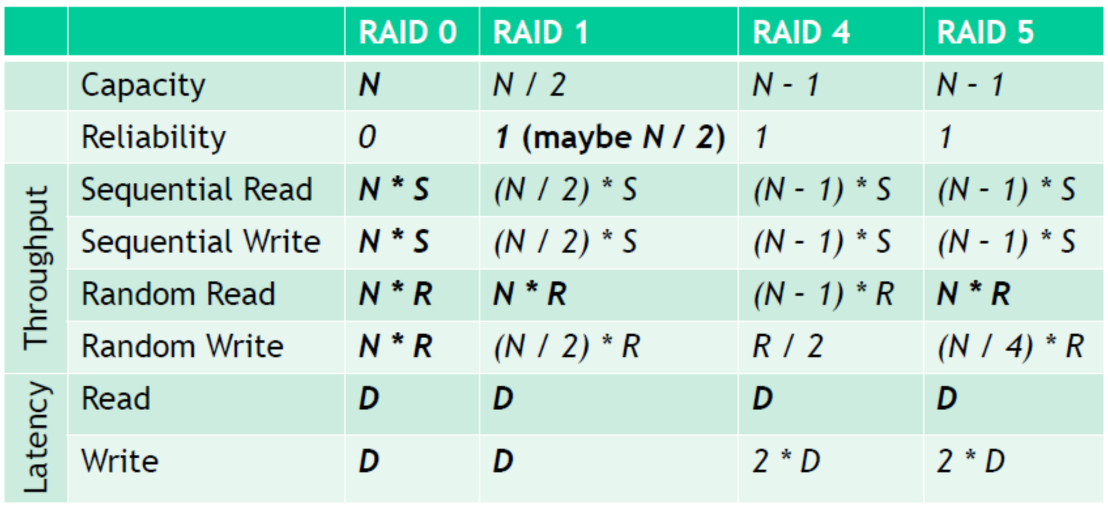
Parity blocks are spread over all N disks.

RAID 6

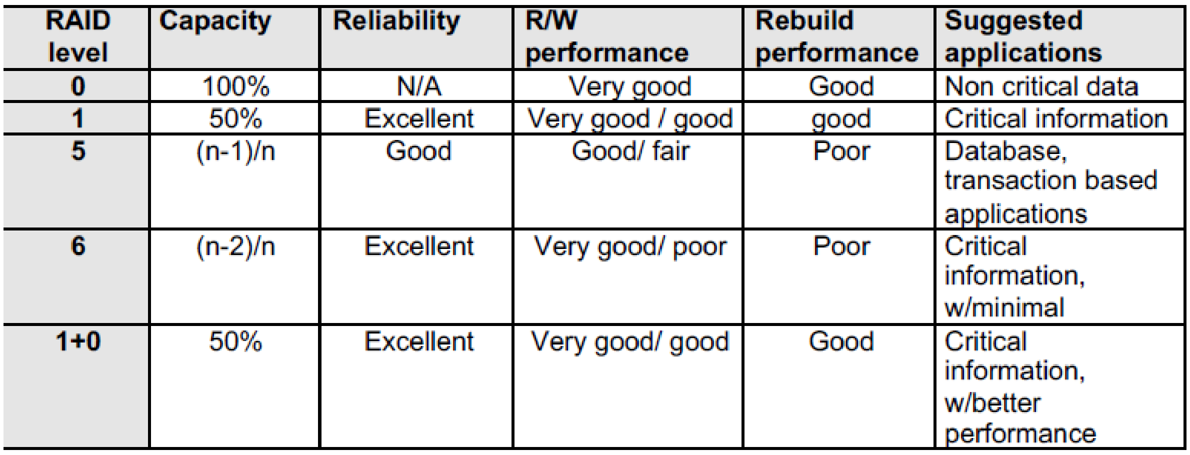
More fault tolerance because we have two parity block per stripe, so 2 concurrent failures are tolerated. It uses Solomon-Reeds codes with two redundancy schemes (P+Q) distributed and independent.

Two disk failures at the same time are tolerated.

RAID comparison



		RAID 0	RAID 1	RAID 4	RAID 5
	Capacity	**N**	*N / 2*	*N - 1*	*N - 1*
	Reliability	*0*	***1* (maybe *N / 2*)**	*1*	*1*
Throughput	Sequential Read	***N * S***	*(N / 2) * S*	*(N - 1) * S*	*(N - 1) * S*
	Sequential Write	***N * S***	*(N / 2) * S*	*(N - 1) * S*	*(N - 1) * S*
	Random Read	***N * R***	***N * R***	*(N - 1) * R*	***N * R***
	Random Write	***N * R***	*(N / 2) * R*	*R / 2*	*(N / 4) * R*
Latency	Read	***D***	***D***	***D***	***D***
	Write	***D***	***D***	*2 * D*	*2 * D*



RAID level	Capacity	Reliability	R/W performance	Rebuild performance	Suggested applications
0	100%	N/A	Very good	Good	Non critical data
1	50%	Excellent	Very good / good	good	Critical information
5	(n-1)/n	Good	Good/ fair	Poor	Database, transaction based applications
6	(n-2)/n	Excellent	Very good/ poor	Poor	Critical information, w/minimal
1+0	50%	Excellent	Very good/ good	Good	Critical information, w/better performance

Many RAID systems include a hot spare, so if a drive fails, the array is immediately rebuilt using the hot spare.

RAID Disks: Reliability Calculation

MTTF = Mean Time To Failure

- RAID 0
  - $MTTF_{RAID\ 0} = MTTF_{single\ disk}/N$
- RAID 1 (per N=2)
  - $MTTF_{RAID\ 1} = (MTTF_{single\ disk}/N) * (1/(MTTR/MTTF_{single\ disk})$
- RAID 1+0
  - $MTTF_{RAID\ 1+0} = (MTTF_{single\ disk}{}^2)/(N * MTTR)$
- RAID 0+1
  - $MTTF_{RAID\ 0+1} = (MTTF_{single\ disk}{}^2)/(N * G * MTTR)$
    - $G = N/2$ is the number of disks in a stripe group
- RAID 4
  - $MTTF_{RAID\ 4} = (MTTF_{single\ disk}{}^2)/(N * N - 1 * MTTR)$
- RAID 5
  - $MTTF_{RAID\ 5} = (MTTF_{single\ disk}{}^2)/(N * N - 1 * MTTR)$
- RAID 6
  - $MTTF_{RAID\ 6} = 2 * (MTTF_{single\ disk}{}^3)/(N * N - 1 * N - 2 * MTTR)$

LESSON – DEPENDABILITY

Dependability

- Attributes
    - Reliability
    - Availability
    - Safety
    - Integrity
    - Maintainability
    - Testability

Reliability

The ability of a system or component to perform its required functions under stated conditions for a specified period of time.

$$R(t)$$

$1 - R(t) \rightarrow$ unreliability also denoted Q(t)

Often used to characterize systems in which even small periods of incorrect behaviour are unacceptable. (ex. impossibility or difficult to repair)

Availability

The degree to which a system or component is operational and accessible when required for use.

Availability = Uptime / (Uptime + Downtime)
$A(t) \rightarrow$ probability that the system will be operational at time t
$1 - A(t) \rightarrow$ unavailability

When the system is not repairable $\rightarrow A(t) = R(t)$
In general, $A(t) \geq R(t)$

MTTF (Mean Time To Failure)
MTBF (Mean Time Between Failures) $= \frac{total\ operating\ time}{number\ of\ failures}\ or\ \frac{1}{\lambda}\ where\ \lambda = \frac{number\ of\ failures}{total\ operating\ time}$

Reliability terminology:

- Fault, a defect within the system
- Error, a deviation from the required operation of the system or subsystem
- Failure, the system fails to perform its required function

Reliability Block Diagram

Every element in the RBD has its own reliability and these blocks are then combined to model all the possible success paths.

- Component in series
    - 2 blocks, C1 and C2

$$R_S = R_{C1} * R_{C2}$$

In general, if system S is composed by components with a reliability having an exponential distribution (very common case)

$$R_s(t) = e^{-\lambda_s t} \; where \; \lambda_s = \sum_{i=1}^{n} \lambda_i \rightarrow MTTF_s = 1/(\sum 1/MTTF_i)$$

A special case: when all components are identical

$$R_s(t) = e^{-\lambda_s t}$$

$$R_S(t) = e^{-n\lambda t} = e^{-\frac{nt}{MTTF_1}} \qquad MTTF_S = \frac{MTTF_1}{n}$$

Availability:

$$A_S = \prod_{i=1}^{n} \frac{MTTF_i}{MTTF_i + MTTR_i}$$

When all components are the same:

$$A_S(t) = A_1(t)^n \qquad A = \left(\frac{MTTF_1}{MTTF_1 + MTTR_1}\right)^n$$

- Component in parallel
  - 2 blocks, C1 and C2

$$R_S = R_{C1} + R_{C2} - R_{C1} * R_{C2}$$

System P composed by *n* components

$$R_P(t) = 1 - \prod_{i=1}^{n} \left(1 - R_i(t)\right)$$

Availability

$$A_P(t) = 1 - \prod_{i=1}^{n} \left(1 - A_i(t)\right)$$

$$A_P = 1 - \prod_{i=1}^{n} \left(1 - A_i\right) = 1 - \prod_{i=1}^{n} \frac{MTTR_i}{MTTF_i + MTTR_i}$$

A system may be composed of two parallel replicas:

- the primary replica working all time, and
- the redundant replica (generally disable) that is activated when the primary replica fails

We need

- a mechanism to determine whether the primary replica is working properly or not
- a dynamic switching mechanism to disable the primary replica and activate the redundant one

LESSON – PERFORMANCE MODELING

Computer performance: the total effectiveness of a computer system, including throughput, individual response time and availability.
Can be characterized by the amount of useful work accomplished by a computer system or computer network compared to the time and resources used.

How can we evaluate system quality?

- Use of intuition and trend extrapolation (rapid but not accurate)
- Experimental evaluation of alternatives (accurate but inflexible and laborious)

We can use a model-based approach since that it is simpler than the actual system, captures the essential characteristics and can be evaluated to make predictions.

3 Techniques:

1. Analytical and Numerical techniques are based on the application of mathematical techniques, which usually exploit results coming from the theory of probability and stochastic process.
2. Simulation techniques are based on the reproduction of traces of the model (they are the most general but also the less accurate).
3. Hybrid techniques, combine the previous two techniques.

Queueing network modelling

It is a particular approach to computer system modelling in which the computer system is represented as a network of queues. A network of queues is a collection of service centres, which represent system resources, and customers, which represent users or transactions.

Different aspects characterize queuing models:

- Arrival
    - It represents jobs entering the system, they specify how fast, how often and which types of jobs does the station service. We are interested in the average arrival rate λ (req/s).
- Service
    - It represents the time a job spends being served.
    - The service time is the time which a server spends satisfying a customer.
    - The important characteristics of this time will be its average duration. If the average duration of a service interaction between a server and a customer is $1/\mu$ where μ is the maximum service rate.
    - Possible situations:
        - A single server (one customer at a time)
        - An infinite server (as many servers as there are customers, so no queueing)

In the real world, we have c servers, if the number of customers is less than or equal to c, there will be no queueing, otherwise, the additional customers will have to wait in the queue.

- Queue

- If jobs exceed the capacity of parallel processing of the system, they are forced to wait queueing in a buffer.
  If the buffer has finite capacity, there are 2 alternatives when it becomes full:
    - Arrivals are suspended
    - Arrivals continue and arriving customers are lost
- Queuing policy determines which of the job in the queue will be selected to start its service.
    - FCFS first come first serve
    - LCFS last come first serve
    - RSS random-selection service
    - PRI priority

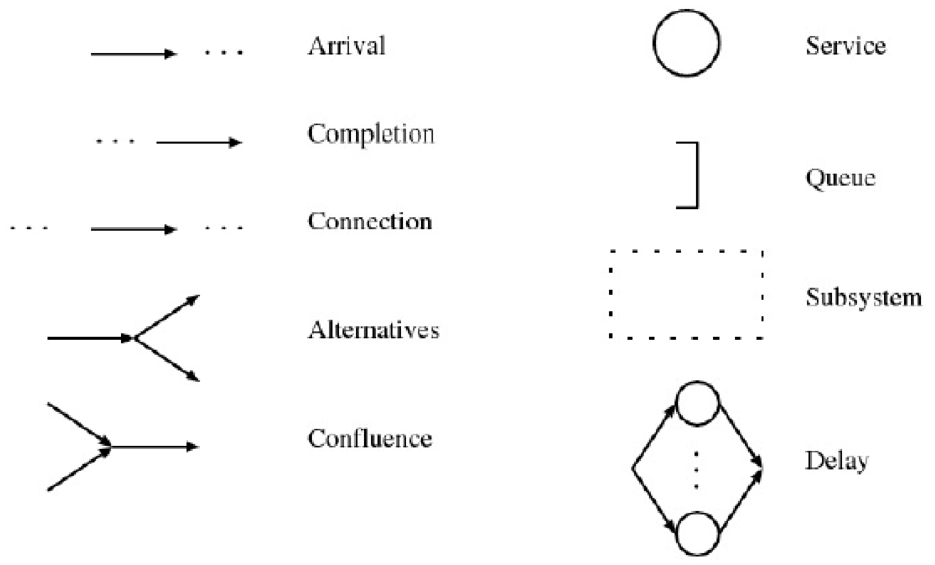
- Population
    - The characteristic of the population which we are interested in is usually the size.
    - The size can be fixed, so N customers will require the service, or can be so large that we can assume it as infinite.
    - We can divide the population into classes whose members all exhibit the same behaviour.

A queueing network can be represented as a graph where nodes represent the service centres k and arcs the possible transitions of users from one service centre to another.

A network can be:

- Open – customers arrive and depart to external environment (ex. Client – Server system)
- Closed – a fixed population of customers (N) remain within the system (ex. Client – Server system in a local network)
- Mixed - there are classes of customers within the system exhibiting open and closed patterns of behaviour respectively

Graphical notation



Routing in queueing network

Whenever a job, after finishing service at a station has several possible alternative routes, an appropriate selection policy must be defined. The policy that describes how the next destination is selected is called *routing*.

The main routing algorithms are:

- Probabilistic
  - each path has assigned a probability of being chosen by the job that left the considered station
- Round robin
  - the destination chosen by the job rotates among all the possible exits
- Join the shortest queue
  - jobs can query the queue length of the possible destinations, and chose to move to the one with the smallest number of jobs waiting to be served

LESSON – OPERATIONAL LAWS

Operational laws are simple equations which may be used as an abstract representation or model of the average behaviour of almost any system.
The laws are very general and make almost no assumptions about the behaviour of the random variables characterising the system.

Operational laws are based on observable variables:

- T, the length of time we observe the system
- A, the number of request arrivals we observe
- C, the number of request completions we observe
- B, the total amount of time during which the system is busy
- N, the average number of jobs in the system

From these observed values, we can derive the following 4 quantities:

1. $\lambda_k = A_k/T$, the arrival rate
2. $X_k = C_k/T$, the throughput or completion rate
3. $U_k = B_k/T$, the utilisation
4. $S_k = B_k/C_k$, the mean service time per completed job

(The upper formulas are also valid if we have total data and not partial ones like $X = C/T$)

If the system is job flow balanced the arrival rate will be the same as the completion rate, that is:

$$\lambda = X$$

A system may be regarded as being made up of several devices or resources. Each of these may be treated as a system from the perspective of operational laws.

Utilization Law	$U_k = X_k * S_k$	
Little Law	$N = X * R$	If the system throughput is X requests/sec, and each request remains in the system on average for R seconds, then for each unit of time, we can observe on average XR requests in the system.
Interactive Response Time Law	$R = (N/X) - Z$	The think time represents the time between processing being completed and the job becoming available as a request again.
Forced Law	$X_k = V_k * X$	The forced flow law captures the relationship between the different components within a system. It states that the

		throughputs or flows, in all parts of a system must be proportional to one another.

When considering nodes characterized by visits different from one, we can define two permanence times:

- Response Time (R'_k), accounts for the average time spent in station k, when the job enters the corresponding node
- Residence Time (R_k), accounts instead for the average time spent by a job at station k during the staying in the system: it can be greater or smaller than the response time depending on the number of visits.

Other operational laws:

$V_k = C_k/C$, visit count of the k-th resource
$D_k = S_k * V_k$, the total amount of service that a system job generates at the k-th resource

LESSON – PERFORMANCE BOUNDS

Provide valuable insight into the primary factors affecting the performance of computer system. (single class systems only)

Determine asymptotic bounds, i.e., upper and lower bounds on a system's performance indices X and R:

- in our case, we will treat *X* and *R* bounds as **functions of number of users** or **arrival rate** (i.e., λ or N)

Advantages:

- highlight and quantify the critical influence of the system *bottleneck*.
- Computed quickly and easy
- Useful in system sizing
- Useful for system upgrades

Notation:

- K, the number of service centres
- D, the sum of the service demands at the centres, so
  - $D = \sum D_k$
- D_{max}, the largest service demand at any single centre
- Z, the average think time, for interactive systems

And the following performance quantities are considered:

- X, the system throughput
- R, the system response time

By considering the (asymptotically) extreme conditions of light and heavy loads:

- *Optimistic*: X upper bound and R lower bound
- *Pessimistic*: X lower bound and R upper bound

Open models

X bound = the maximum arrival rate that the system can process

If $\lambda > X$ bound → the system SATURATES → new jobs have to wait an indefinitely long time. The X bound is calculated as: $\lambda_{sat} = 1/D_{max}$

R bound = the largest and smallest possible R experienced at a given λ investigated only when $\lambda < \lambda_{sat}$(otherwise the system is unstable!)

Bound for R(λ): $R(\lambda) \geq D$

2 extreme situations:

1. No customers → no queue, then $R = D$
2. There is no pessimistic bound on R:

a. if *n* customers arrives together every n/λ time units
(the system arrival rate is $n/(n/\lambda) = \lambda$
b. customers at the end of the batch are forced to queue for customers at the front of the batch, and thus experience large response times
c. as the batch size n increases, more and more customers are waiting an increasingly long time
d. thus, for any postulated pessimistic bound on response times for system arrival rate λ, it is possible to pick a batch size n sufficiently large that the bound is exceeded

Closed models

X bounds considered first, then converted in R bounds using Little's Law.

Light Load situation (lower bounds): 1 customer case

$$N = X * (R + Z) \;\&\; 1 = X * (D + Z) \; then \; X = 1/(D + Z)$$

(adding customers)

In closed models, the **highest possible system response time** occurs when each job, at each station, founds all the other N-1 costumers in front of it.

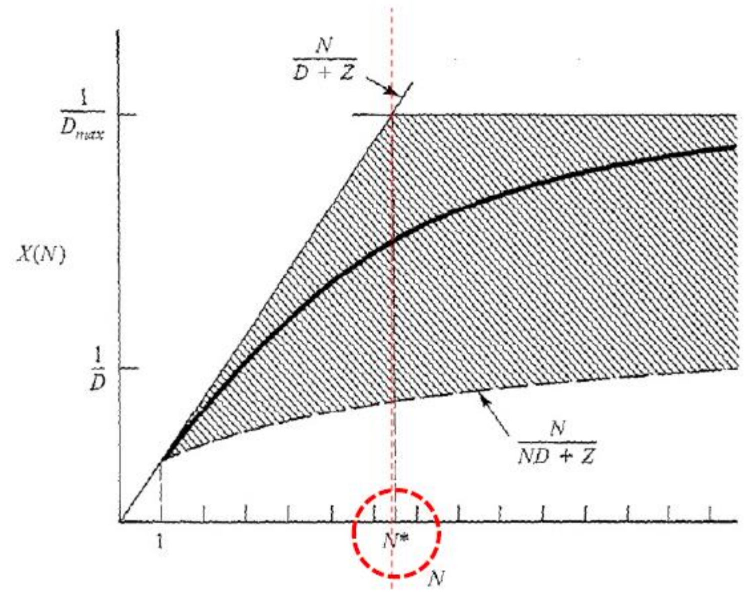
In this case the $X = N/(ND + Z)$

The **lowest response time** can be obtained if a job always finds the queue empty and always starts being served immediately.

In this case the $X = N/(D + Z)$

Heavy Load situation (upper bound):

Since the first to saturate is the bottleneck (max) → $X(N) \leq 1/D_{max}$



Particular population size determining if the light or the heavy load optimistic bound is to be applied

$$N^* = (D + Z)/D_{max}$$

Bound for closed models:

$$\begin{aligned} \frac{N}{ND + Z} \leq \quad & X(N) \quad \leq \min\left(\frac{N}{D+Z}, \frac{1}{D_{max}}\right) \\ \max\left(D, ND_{max} - Z\right) \leq \quad & R(N) \quad \leq ND \end{aligned}$$